# Routing End User Queries to Enterprise Databases

**Saikrishna Sudarshan**[1] **Tanay Kulkarni**[1] **Manasi Patwardhan**[2]
**Lovekesh Vig**[3] **Ashwin Srinivasan**[4] **Tanmay Tulsidas Verlekar**[4]

[1]BITS Pilani KK Birla Goa Campus, Goa, India
[2]TCS Research, Pune, India [3]TCS Research, Delhi, India
[4]Dept. of CSIS, BITS Pilani KK Birla Goa Campus, Goa, India

{f20211645,f20212615}@goa.bits-pilani.ac.in
{manasi.patwardhan,lovekesh.vig}@tcs.com
{ashwin,tanmayv}@goa.bits-pilani.ac.in

## Abstract

We address the task of routing natural language queries in multi-database enterprise environments. We construct realistic benchmarks by extending existing NL-to-SQL datasets. Our study shows that routing becomes increasingly challenging with larger, domain-overlapping DB repositories and ambiguous queries, motivating the need for more structured and robust reasoning-based solutions. By explicitly modelling schema coverage, structural connectivity, and fine-grained semantic alignment, the proposed modular, reasoning-driven re-ranking strategy consistently outperforms embedding-only and direct LLM-prompting baselines across all the metrics.

## 1 Introduction

Large enterprises operate across verticals, resulting in the enterprise data being distributed across a variety of heterogeneous sources, such as knowledge graphs, relational databases, document repositories, etc. In the context of enterprise-level search, when an end-user submits a query in natural language (NL), it becomes essential to accurately route the query to the most relevant data sources that are capable of providing a correct and comprehensive response. Motivated by this practical challenge, Mandal et al. (2025) introduce a novel task of routing NL queries to appropriate data sources, specifically focusing on enterprise databases (DBs) as data sources, which encapsulate information across diverse domains.

Mandal et al. (2025) extend existing datasets developed for cross-domain NL-to-SQL semantic parsing (Yu et al., 2018; Li et al., 2024) to construct benchmarks for the novel task of query routing, and accordingly define only in-domain and cross-domain settings. However, by preserving the original DB splits, where the test sets contain DBs only from the test split of the underlying datasets, the resulting repository size is extremely small and uneven across settings. This leads to a skewed distribution of DBs and queries in the in-domain and cross-domain scenarios, causing an imbalance in their test sets. Consequently, the cross-domain setting exhibits disproportionately higher recall and mean average precision (mAP) compared to the in-domain setting, which is counter-intuitive and highlights limitations of the benchmark in fairly evaluating routing performance under realistic data distributions.

In enterprise environments, multiple DBs frequently overlap in domain coverage, leading to shared table names, column names, and values, which significantly increases the difficulty of accurate query routing. Motivated by this and to address the above limitations, we construct a more realistic benchmark (Section 3). We merge all DBs from both the train and test splits of the original datasets (Yu et al., 2018; Li et al., 2024) to form a unified DB repository and perform a 50–50% random split of the **queries** within each DB into train and test queries. We provide both train and test splits to support future work that may require supervised training, although our proposed method itself is training-free, as DB schemas are inherently dynamic and frequent fine-tuning is impractical. In the spirit of prior work, we define two settings: (i) **In-domain**, where queries corresponding to every DB are observed during both training and testing, and the inference repository consists of *all* DBs, unlike prior work where the training repository contained only train DBs and the test repository contained only test DBs; and (ii) **Cross-domain**, where queries of a subset of DBs are completely unseen during training and are introduced only at inference time, while keeping the DB repository unchanged.



*Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026)*, pages 260–268

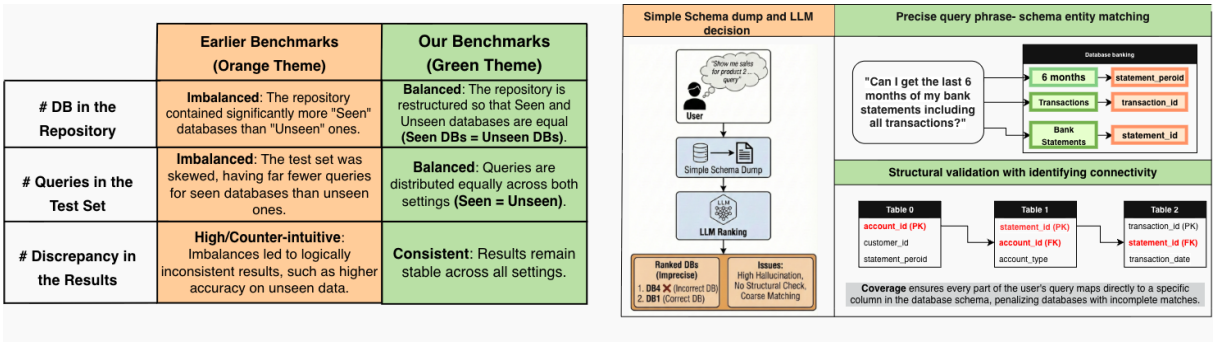


Figure 1: Limitations of existing benchmarks and approaches addressed by our benchmark and approach

*Note that, in either setting, a query never appears in both the training and test sets, since a 50–50 partition of queries is performed prior to defining the experimental splits.* The in-domain setting evaluates routing when query patterns of the target DBs are already observed, whereas the cross-domain setting models a realistic enterprise scenario in which DBs are changed and **new** queries must be routed to previously unseen DBs. This formulation preserves an identical repository across both settings and enables a fair and meaningful comparison.

We re-implement the techniques of Mandal et al. (2025) as baselines on our benchmark. The inferior performance observed relative to their original results demonstrates the increased difficulty and realism of our setting. We further propose a novel training-free method that combines Schema Entity Recognition with Large Language Models (LLMs) to achieve state-of-the-art performance on this challenging benchmark. The main contributions of this work are as follows:

- We develop more realistic and robust benchmarks for the DB routing task defined in (Mandal et al., 2025) by extending two text-to-SQL datasets, viz, Spider (Yu et al., 2018) and Bird-SQL (Li et al., 2024).
- We define a new technique for the task that yields state-of-the-art results on the benchmark.
- We perform a detailed qualitative analysis of our results, highlighting the efficacy of our technique.

## 2 Problem Statement

We have an end-user question $q$ and a repository of a set of DBs $D$, where each DB is indexed by a DB id $d_x$ with schema $S_x$ consisting of multiple tables and columns ($\{T_x\}$, $\{C_{xy}\}$), where $\{T_x\}$ is the set of tables that belong to the DB schema $S_x$ and $\{C_{xy}\}$ is the set of columns of tables $\{T_x\}$. The task is to rank the DBs in $D$ for the question $q$ based on their relevance in terms of answerability (whether the DB can provide the correct answer). We have train and test set queries $\{q_{x_{tr}}\}$ and $\{q_{x_{te}}\}$ for each DB $d_x$ such that $\{q_{x_{tr}}\} \cap \{q_{x_{te}}\} = \phi$.

| Metric | Spider route | BIRD route |
|---|---|---|
| Total Number of DBs | 206 | 80 |
| Total Questions | 11,831 | 10,962 |
| Train Questions | 5,892 | 5,461 |
| Test Questions | 5,939 | 5,501 |

Table 1: *Spider* and *BIRD* Dataset Statistics

## 3 Dataset Construction

We extend two existing datasets constructed for cross-domain NL-to-SQL, viz., Spider (Yu et al., 2018) and BIRD-SQL (Li et al., 2024). We name the extended datasets as *Spider-Route* and *Bird-Route*, respectively. The dataset statistics are provided in Table 1.

**Spider-Route** A sample in the original Spider dataset (Yu et al., 2018) consists of a DB, a NL question posed on the DB and the corresponding SQL query. Each DB is associated with comprehensive schema information, including table and column names, data types, and primary-foreign relationships, which is defined through SQL Data Definition Language (DDL) scripts. The DBs in the train and test split of the original dataset are distinct. For *Spider-Route*, we combine the DBs to form a repository of 206 DBs. We formulate 50-50% split of the questions belonging to each DB to form train and test splits for *Spider-Route*, ensuring that there is no overlap between the questions. This leads us to a total number of $5,892$ training and

5, 939 test queries.

**Bird-Route** The BirdSQL (Li et al., 2024) dataset is a collection of DBs spanning over a diverse set of domains such as medical, finance, education and sports. A question in the dataset is accompanied by domain knowledge, termed as 'evidence', which can be used to resolve the query to SQL. A sample in the original Bird-SQL consists of a DB, a question in NL posed on the DB, question-specific evidence, and the corresponding SQL query. For each DB $d_x$, we take the union of all the question specific evidences for all the queries which belong to that DB to form DB level meta-data $m_x$. The meta-data varies from 2 to 48 sets of evidence sentences per table in the DB. We formulate *Bird-Route*, by combining the DBs of the original train (69) and dev splits (11) of BirdSQL to form the DB corpus $D$ of 80 DBs and perform 50-50% split of questions of each DBs to form train and test splits of 5461 and 5501 queries, respectively.

**Database: `authors_schema`: Table: Author**
`CREATE TABLE "Author" (Id INTEGER ... Name TEXT, Affiliation TEXT)`
Table Description: Author
**COLUMN 1** column_name: Id ...
**COLUMN 3** column_name: Affiliation; Column description: Organization name with which the author is affiliated.; Data format: text

Table 2: Example Schema Format (DDL)

You are an expert DB Administrator tasked with evaluating and refining the relevance ranking of DBs for specific questions. You will be provided with a question and an initial ranking of the top 5 DBs deemed most relevant. Along with this you will be provided the schemas of these DBs. Your task is to analyze the question, critically assess the initial ranking, and provide a revised ranking of the top 3 DBs, where the DB at Rank 1 is *absolutely the most relevant* to answering the given question.
**Input:** Question [ID]: Text: [Question text]; Ranking (Top 5): Rank 1-5: [DB_1-5]; Schemas: [Schema 1-5]
**Output:** Q [ID]: [DB_1], [DB_2], [DB_3]
**Rules:** Rank 1 must be the *undisputed* best match. All 3 DBs must be distinct and from the initial ranking. Do not violate the output format or add extra text.

Table 3: Prompt for LLM Re-Ranking (Top-5 DBs)

## 4 Approach

In this section we elaborate on our novel technique to perform the ranking task discussed in Section 2. For this technique, we use pre-trained models and do not perform any task specific fine-tuning. Thus, our method is invariant to in-domain and cross-domain settings. For each question we first compute the cosine similarity between the question embedding and the schema embeddings of the DBs in the repository. The DB schema is defined following the DDL script (Refer Table 2) Along with the schema definition for the BIRD-Route dataset we use the meta-data in the form of column and value descriptions (discussed in Section 3) for richer schema representation. The embeddings in this case are generated using the gte-Qwen2-7B-instruct (Yang et al., 2024), yielding the best results. The DBs are ranked according to the similarity scores. The top-k DBs with the highest similarity score are then selected for the re-ranking step.

Following Mandal et al. (2025), we perform re-ranking by prompting a Large Language Model (LLM) (Gemini 2.0 Flash, part of the Gemini model family (Comanici et al., 2025), in our case) with the top-k DB schema, their meta-data, along with the query. The prompt is reported in Table 3.

We observe that a direct LLM call tends to erroneously associate query elements with the DB elements based on lexical similarity rather than structural intent of the query, leading to erroneous DB ranking. For example, for the query: *Find the attribute data type for the attribute named 'Green'*, the DB `products_gen_characteristics` is incorrectly ranked higher than the ground-truth DB `product_catalog`. This error arises because the LLM implicitly interprets the token '*Green*' as a value of the **Colour** field of `products_gen_characteristics` DB. However, as specified in the query, '*Green*' refers to an **attribute name** itself and not a field value of the attribute **Colour**.

We address this limitation by proposing a novel re-ranking strategy, which allows LLMs to associate the queries more accurately with the DB schema. We decompose the problem into modular sub-tasks, allowing LLMs to focus on simplified, localised and well-scoped reasoning, rather than full end-to-end schema-query alignment. Given top-k candidate DBs and the query, we compute a 'total score' and a 'semantic score' for each DB for that query. We use the 'total score' to re-rank the DBs for the given query. For a DB $d_x$ with schema $S_x$ and a NL query $q$, the total score indicates how well the DB $d_x$ can answer the query $q$, with 1.0 being fully appropriate and 0.0 being irrelevant. The 'semantic score' is used for tie-breaking when two DBs receive the same 'total score'. To compute the 'total score' we follow the following steps:

**Step 1: Construction of Adjacency List:** The

step determines the joins that can be formed between the tables in a DB based on the schema. The LLM is prompted (prompt in Appendix A.) to produce an adjacency list representing possible joins between the tables and corresponding column names, by providing the schema and table, column descriptions as the input. For example, consider a DB: *Activity*, containing the following five tables with corresponding columns: **Activity**(`activity_id`, `activity_name`); **Participates_in**(`student_id`, `activity_id`); **Faculty_Participates_in**( `faculty_id`, `activity_id`); **Student**(`student_name`, `student_id`); and **Faculty**( `faculty_name`, `faculty_id`) The LLM is supposed to produce an adjacency list depicting possible joins amongst the tables in the schema of the form:

$$\{0 : \{1, 2\};\ 1 : \{0, 3\};\ 2 : \{0, 4\};\ 3 : \{1\};\ 4 : \{2\}\}$$

This adjacency list is used for connectivity checks in the following steps. Since this step can be performed independently of the query, it is executed just once for each DB.

**Step 2: Query Phrase-Schema Entity Mapping:** In this step, the LLM is prompted to extract the relevant query spans that can be mapped to column names and values in a DB. Specifically, the prompt (Appendix B.) requires the LLM to not only perform named entity recognition for the given query, but also understand the intent and semantics of the query in the context of the DB schema to identify the relevant query spans. LLM is prompted to identify not only explicit noun phrases but also verbs, adjectives, or expressions (such as average or sum), which implicitly refer to the entities in the schema. For example, for the query *'What does John do?'*, the LLM is not only supposed to identify the noun *'John'* to map it to an equivalent value in column `Student.student_name` of the **Activity** DB, but also needs to identify the verb *'do'* to be interpreted as referring to a person's activity and thus semantically maps to the column `Activity.activity_name` of the DB. The query phrase identification in this context is a complex task involving both syntactic parsing and semantic associations. For a given query, this step is run for each DB, the output is a set of query phrase-to-DB schema entity mappings, where one query-phrase might be mapped to multiple DB-schema entities. For example, for the above query, the extracted phrases and entity mappings for DB Activity are:

`John` → `Student.student_name`
`John` → `Faculty.faculty_name`
`do` → `Activity.activity_name`

If an identified phrase cannot be matched to any entity, it is marked as `N/A`. While in the case of a phrase with multiple mappings, all the mappings are considered valid.

**Step 3: Entity Coverage and Table Connectivity:** To assess the answerability of a query on a DB we require the query - DB pair to satisfy two criteria: Phrase coverage and Connectivity. Phrase Coverage requires all phrases to be mapped to valid DB columns, i.e., no `N/A` mappings. A coverage score is assigned to the query DB pair according to the number of unique successfully mapped phrases. Unmapped phrases exponentially decrease the total score to penalise missing phrases harshly. Entities with multiple mappings are counted only once. The coverage score is computed as:

$$coverage = e^{-nx} \tag{1}$$

$$x = \frac{N_{\text{NA}}}{N_{\text{total}}}, \quad x \in [0, 1]. \tag{2}$$

where $N_{\text{NA}}$ is the number of `N/A` mappings, $N_{\text{total}}$ is the total number of mappings, and $n \in [1, +\infty)$ is a hyperparameter controlling the penalty. Small $n$ gives a softer penalty, while large $n$ penalizes `N/A` maps more harshly.

Connectivity requires that all schema entities mapped from the query phrases form a connected subgraph in the database schema adjacency list. Since each phrase may admit multiple candidate mappings, the connectivity constraint is satisfied if at least one combination of mappings yields a connected component. Owing to the availability of the schema adjacency list, this can be efficiently verified using a simple graph traversal algorithm such as *Breadth-First Search (BFS)* to test reachability among the selected entities. In our example, the mappings for the phrases `'John'` and `'do'` must be mutually reachable in the schema graph. There exist two valid paths that satisfy this criterion:

`Table 3` → `Table 1` → `Table 0` (mapping `John` as a student)
`Table 4` → `Table 2` → `Table 0` (mapping `John` as a faculty member)

Since at least one valid connected subgraph exists, the connectivity criterion is met, and hence the connectivity score for this query–database pair is

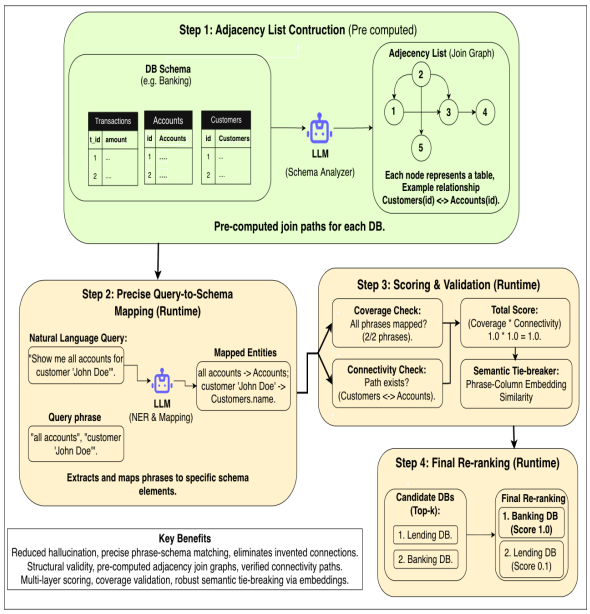


Figure 2: Our Approach: Modular Reasoning Re-Ranking

assigned a value of 1. If this is not the case, a score of 0 is assigned, immediately invalidating the database.

**Step 4: Total and Semantic Score Computation:** The total score can then be computed as the product of coverage and connectivity:

$$total\_score = coverage \times connectivity \quad (3)$$

We then calculate the Semantic Score as follows: For each mapped query phrase, we calculate the cosine similarity between the embeddings of the query phrase and its matched column name and information. This is done for all mapped query phrases with corresponding mapped schema entities and the semantic score is the average of all the cosine similarity scores and is used for tie-breaking when two DBs receive the same 'total score'.

Thus, the proposed method ensures that LLMs are employed judiciously for simpler subtasks where their reasoning is reliable, while structural aspects like table connectivity and coverage are handled algorithmically. This method achieves more robust and explainable DB ranking in the face of ambiguous or underspecified NL queries.

## 5 Results and Discussion

**Metric:** For each question, we calculate the recall@1 (R1), recall@3 (R3) and mean average precision (mAP). For a question, Recall@k is 1 if the DB ranked in the top-k by an approach is the ground truth DB and is 0 otherwise.

**Baselines:** We use the following approaches defined in (Mandal et al., 2025) as our baseline approaches: (i) all-mpnet-base-v2 (Reimers and Gurevych, 2019) embeddings (**B1**): We embed the question as well as DB schema using all-mpnet-base-v2 (Reimers and Gurevych, 2019) model. For a given question we find cosine similarity of its embeddings with the embeddings of all DBs and use this similarity as a scoring function to rank all the DBs for that question. (ii) LLM Re-ranking (**B2**): We use the top-k DBs identified by the embedding based method and use the prompt defined in (Mandal et al., 2025) to employ LLMS (Gemini (Comanici et al., 2025)) re-rank those DBs for the given question. Consequently, the recall@1 for this approach is inherently limited by the recall@5 of (**B1**). We also use gte-Qwen2-7B-instruct (Yang et al., 2024) embedding model to get results of (i) (**B3**) and (ii) (**B4**) above, which serves as a better embedding model for us. We replace the LLM based re-ranking method with our re-ranking method and use top-k DBs identified by the gte-Qwen2-7B-instruct embedding based method for re-ranking. For our experimentation we use the value of k to be 5.

The results on *Spider-Route* and *Bird-Route* are discussed in Table 4. We also answer the following research questions:

**Q1: Is the newly created benchmark more robust than the prior benchmark for the DB Routing task?** As discussed in section 3, as opposed to the approach in Mandal et al. (2025), we recreate the dataset leading to consistent number of DBs in the repository across in-domain and cross-domain settings. Overall, the number of DBs in the repository is larger in our benchmark as compared to the prior one, making the task harder and more realistic. This is also demonstrated by the performance on the prior and our benchmark with the baseline approach **B1**. **B1** leads to recall@1 of 44.09 on *Spider-Route* and 80.21 on the *BIRD-Route* datasets of Mandal et al. (2025), respectively. Whereas, recall@1 of the same method **B1** on our *Spider-Route* and *BIRD-Route* datasets is 47.47 and 43.65 , respectively (Table 4). The proposed organisation for the *Spider-Route* and *BIRD-Route* dataset in this work, having more number of DBs in the repository creates a harder setting to differentiate amongst the DBs for a query, especially the ones which are closer with respect to the semantics of its schema.

**Q2: Does similarity in domains of the data-**

| Method | BIRD-Route Dataset | | | | Spider-Route Dataset | | | |
|---|---|---|---|---|---|---|---|---|
| | R@1 | R@2 | R@3 | MAP | R@1 | R@2 | R@3 | MAP |
| **B1:** Cosine with all-mpnet-base-v2 (Reimers and Gurevych, 2019) embedding | 0.4747 | 0.5733 | 0.6774 | 0.5751 | 0.4365 | 0.5191 | 0.6288 | 0.5281 |
| **B2:** LLM Reranking (all-mpnet-base-v2 + Llama 3 (Grattafiori et al., 2024)) | 0.5429 | 0.5971 | 0.7286 | 0.6229 | 0.4941 | 0.6325 | 0.6512 | 0.5926 |
| **B3:** Cosine with gte-Qwen2-7B-instruct (Yang et al., 2024) embedding | 0.6249 | 0.7821 | 0.8344 | 0.7471 | 0.6033 | 0.6835 | 0.7471 | 0.6780 |
| **B4:** LLM Reranking with Gemini (Comanici et al., 2025) | 0.7120 | 0.8060 | 0.8490 | 0.7890 | 0.6799 | 0.7166 | 0.7613 | 0.7193 |
| **Ours:** LLM Re-ranking with Modular Reasoning | **0.7962** | **0.8176** | **0.8491** | **0.8210** | **0.7865** | **0.7942** | **0.8086** | **0.7964** |

Table 4: Results on BIRD-Route and Spider-Route Datasets

**sources affect the performance of the DB-Routing?** We first encode the DDL schema definitions of the DBs using the gte-Qwen2-7B-instruct (Yang et al., 2024) embedding and then cluster using a constrained K-means algorithm grouping semantically similar DBs. A size constraint is enforced, ensuring even distribution of DBs across clusters, such that each cluster belongs to a domain. The DBs of *Spider-Route* result in 10 clusters containing 20 to 23 DBs each, whereas DBs of *Bird-Route* result in 5 clusters containing 15 to 17 DBs each. We observe that for 82% of questions for *Spider-Route* and 76% of questions of *Bird-Route*, the top-5 ranked DBs belong to the same cluster and thus same domain. Thus, for the DB-Routing task DB selection is hardest within the same domain.

To quantify in-cluster confusion in our semantic embedding-based approach, we analyze all cases where the correct DB is **not** ranked at top-1. Among these errors, 86.2% and 75.0% of questions in *Spider-Route* and *Bird-Route* have the top-1 DB from the same domain cluster as the ground truth. Further, 82.3% and 76.2% of questions in *Spider-Route* and *Bird-Route* contain multiple same-domain DBs within the top-5 results. These statistics reveal that semantic embedding errors are predominantly concentrated within the similar semantic domains rather than across unrelated domains, leading to higher intra-cluster error rate. For example, in the *Spider-Route* dataset, when routing fails for book-related queries, the system typically confuses between `book_2`, `book_press`, and `book_review` DBs rather than incorrectly selecting unrelated DBs from sports or healthcare domains. With higher intra-cluster error rate, the *Spider-Route* dataset demonstrates more cluster cohesion (tighter within-domain grouping) and relatively clear decoupling (inter cluster domain separation) when compared to the *Bird-Route*. This highlights the need for a better approach which improves within-domain discrimination capabilities.

**Q3: Does our approach effectively address the challenge of discriminating between highly similar DBs in the DB Routing task?** The proposed method outperforms the direct LLM-based re-ranking method for the DB routing task (Table 4). For the *Spider-Route*, LLM-based re-ranking and the proposed method have similar Recall@3. The results suggest that the direct LLM-based re-ranking is capable of identifying relevant DBs but cannot select the most appropriate DB for a given query. Whereas, our proposed method can distinguish between most similar DBs to identify the right DB for the query. This is due to several key factors that enable more accurate DB re-ranking, particularly in scenarios involving multiple valid candidate DBs, precise entity matching requirements, and complex join operations. The improvement can be attributed to the three main reasons:

(i) ***Precise query phrase-schema entity matching:*** The second step of our approach takes into consideration the coverage of query phrases by the schema for scoring for the re-ranking task. This ensures a more fine-grained similarity score as opposed to a coarse grained similarity check between the complete query and the schema. For example, consider the query discussed earlier: *Find the attribute data type for the attribute named 'Green'*, the proposed method correctly identifies `product_catalog` as the DB with the total score of 1.0 due to its proper attribute structure. The other top ranked DBs `products_gen_characteristics` and `phone_1` receive the total score of 0.188875 and 0, respectively. In contrast to this the baseline LLM re-ranker incorrectly gives the following top-3 ranks: 1.`products_gen_characteristics`, 2.`product_catalog` and 3.`phone_1`, ranking an inappropriate DB first.

(ii) ***Structural validation with identifying connectivity between the schema entities mapped to query phrases:*** Steps 1 and 3 allow taking into consideration the connectivity of the schema entities mapped to the query phrases for the re-ranking score com-

putation. The DB with perfect connectivity score provides structural validation for the query. It ensures that the downstream retrieval of the data from the DB would be possible by writing equivalent SQL for the given query, allowing joining the tables involved in the query. For example, for the query *'What is the transmitter of the radio with the largest ERP_kW?'*, our method correctly identifies `tv_shows` as the top DB with a partial match of score 0.8265. Whereas for other top ranked DBs ( `program_share`, `bbc_channels`, `geo`), we observe that the DB entities mapped to the query phrases are distributed across tables that cannot be joined, leading to a connectivity score of 0. Since $total\_score = coverage \times connectivity$ these DBs are ranked lower than the correct DB. Whereas, the baseline LLM re-ranker incorrectly performs the following top-3 re-ranking: `geo`, `bbc_channels`, `program_share`, selecting DBs with no joining paths between the mapped DB entities.

(iii) ***Effective handling of multiple valid candidates through semantic tie-breaking:*** This step allows fine-grained similarity computations between the mapped query phrases and schema entities, allowing distinguishing between semantically similar DB schemas with effective tie-breaking mechanism. In contrast, the LLM reranker fails to make such fine-grained comparisons between top-k DBs. For example, for the query *'How many books are there?'* with `book_2` being the gold DB on which the query is posed. The baseline LLM re-ranker provides following DBs as the top-3 re-ranked list: `book_review`, `book_press`, `book_2`. However, our method returns the following as the re-ranked list: `book_2`, `book_review`, `book_press`, with total score of all the DBs being 1, leading to the case of a tie. This indicates that all the above DBs are capable of addressing this query. However, our method results in the `book_2` DB obtaining the highest semantic score of **0.688**. As mentioned prior, the semantic score is computed based on the embedding similarity between the query phrase and the corresponding DB entity along with its originating table. In this case, the query term *book* aligns most strongly with `books.BookID` from the `books` table in the `book_2` DB, thereby surpassing alternative but weaker alignments such as `books.ISBN` from the `book_1` DB and `book_press.bookID` from the `book_press` DB.

***Error Analysis:*** Though our approach demonstrates improvement in the scores still there are 20% and 21% queries in *Spider-Route* and *Bird-Route* (Table 4), which do not have 100% Recall@1. We perform error analysis of these queries. 11% and 16% of these queries, respectively, do not have the correct DBs in the top-5 results of the embedding based ranking method `B3`. For such queries we add the correct DB in top-5 and run our re-ranking method. We observe that for 98% and 96% of such queries for *Spider-Route* and *Bird-Route*, even though the DB schema has the lower cosine similarity score with the query embeddings, the proposed method is able to rank it as the top-1 for the query. This indicates that if the right DB is available for re-ranking, our method can accurately identify that DB to be the most relevant one. We perform one more experiment on a subset of 500 erroneous queries of *Spider-Route* and *Bird-Route*. We use Gemini 2.5 (Comanici et al., 2025) as opposed to Gemini 2.0 as the base LLM and use top-20 candidate DBs from the rankings provided by `B3`. On these 500 erroneous questions with our method we get recall@1 as 92% and 89% on *Spider-Route* and *Bird-Route*. This demonstrates that use of better LLM and choosing higher k for top-k ranked DBs, our approach leads us to substantial improvements in the results. We performed this study on a subset due to much higher cost of Gemini 2.5.

For the remaining 9% and 5% failed queries of *Spider-Route* and *Bird-Route*, it is observed that the queries are ambiguous. We find that the DB selected by our method as the top-1 can also be a possible DB which can answer the query. As our benchmark uses queries, originally designed for DB specific NL to SQL task, the queries have inherent ambiguities with regards to answerability with respect to multiple DBs. They can be answered with multiple DBs. For example, for the query 'How many customers in state of CA?' the ground truth DB is `store_1`, whereas our method returns `loan_1` as the top-1, and the ground truth DB is ranked $3^{rd}$. Both DBs contain customer information with state and location attributes and can legitimately answer this query.

This demonstrates that our re-ranking approach identifies correct DBs for ∼80% of the queries. The remaining 20% queries either (i) can be resolved by using higher k value to choose top-k ranked DBs for re-ranking and by using more reliable base LLM, or (ii) are ambiguous, where the top ranked DB by our method is also a valid DB to answer the query along with the ground truth DB. Thus, our method provides a good solution for the DB-

Routing task.

**Q4: Does domain-specific knowledge facilitate improving the performance of the DB-Routing task?** Along with the DB schema, for the *Bird-Route* dataset the input for the DB Routing contains the meta-data. We perform an experiment without inclusion of this meta-data and observe that there is 5% decrease in Recall@1 and 7% decrease in Recall@3. The domain-specific knowledge in the form of meta-data contains information about the purpose of tables, the meaning of ambiguous or polysemous columns. Thus, domain-specific knowledge provides additional semantic context to route queries in scenarios where DDL script of the schema alone cannot provide sufficient context to resolve them, leading to better performance.

## 6 Conclusion

This work introduces a principled framework for routing natural language queries to enterprise databases by combining more realistic benchmarks with a modular, reasoning-driven re-ranking strategy. By explicitly modeling schema coverage, structural connectivity, and fine-grained semantic alignment, the proposed method consistently outperforms embedding-only and direct LLM-based baselines, especially in challenging, closely similar database scenarios. These results highlight that decomposing query routing into interpretable and verifiable sub-tasks is crucial for building accurate, scalable, and reliable enterprise DB routing systems.

## Limitations

Our work has some limitations worth noting. First, the benchmarks are constructed by extending existing NL-to-SQL datasets, whose queries were originally designed for single-DB settings. As acknowledged in our error analysis, this means many queries are inherently ambiguous with respect to multi-DB answerability, meaning they can be legitimately answered by more than one database, yet only a single ground-truth DB is annotated. This makes evaluation noisy, as a "wrong" top-1 prediction may in fact be a valid routing target. Additionally, our method relies on an initial top-k shortlist from embedding-based retrieval, and as observed in our results, a fraction of failed queries (11% on Spider-Route and 16% on Bird-Route) stem from the correct DB not appearing in the top-5 candidates, placing an upper bound on re-ranking performance. Our experiments with a higher k value and a stronger LLM show that this gap is largely recoverable, suggesting a straightforward path for future improvement. Furthermore, the modular re-ranking pipeline depends on LLM outputs for subtasks, since LLMs are inherently stochastic, repeated runs on the same input may yield slightly different mappings or scores, potentially introducing variability in the final rankings.

## References


Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, and Noveen Sachdeva. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aieleen Letman, Akhil Mathur, Alan Schelten, and Amy Yang. 2024. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Li, Reynold Huang, and Zhanwei Lan. 2024. Can LLM already serve as a database interface? a BIg bench for large-scale database grounded text-to-SQLs. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*.

Priyangshu Mandal, Manasi Patwardhan, Mayur Patidar, and Lovekesh Vig. 2025. Dbrouting: Routing end user queries to databases for answerability. *arXiv preprint arXiv:2501.16220*.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing

Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921. Association for Computational Linguistics.

## Appendix A. Adjacency List Construction Prompt

**Role:** You are a Senior Database Architect. Your expertise lies in analyzing relational database schemas to understand data flow, dependencies, and potential join paths based on structure, naming conventions, and explicit constraints.
**Objective:** Analyze the provided database schema and generate a directed adjacency list representing potential direct JOIN relationships between tables (ONLY INNER JOINS). The output must strictly use numerical indices for tables and adhere precisely to the specified format, reflecting both explicit and implied relationships suitable for join operations.
**Input Schema:** [Schema Information] (Assume the schema contains definitions for tables, including column names and types, primary keys (PK), and foreign keys (FK).)
**Analysis and Processing Steps:**

1. **Table Indexing:** Assign a unique, sequential, 0-based numerical index to each table defined in the schema based on the order they appear or are logically grouped.

2. **Relationship Identification (Apply in order of priority):** For each Table A (with index idx_A): Identify likely Primary Key(s) for all tables. Assume standard PK naming conventions (e.g., `id`, `uuid`, or `<table_name>_id`) and PK constraints if specified.
    - *Priority 1: Explicit Foreign Keys (FKs):* Examine explicit FK constraints defined for Table A. If Table A has an FK column (e.g., `b_id`) that references the PK of Table B (with index idx_B), establish a directed relationship idx_A → idx_B.
    - *Priority 2: Naming Conventions & Data Type Matching (Implied FKs):* Examine columns in Table A that are not explicitly defined as FKs but follow common FK naming patterns (e.g., `<ref_table>_id`). If such a column name pattern in Table A suggests a reference to Table B and the data types are compatible, establish idx_A → idx_B.
    - *Priority 3: Simple Name/Type Match (Lower Confidence):* If a column in Table A has the exact same data type as the likely PK of Table B and the name implies a relationship, establish idx_A → idx_B. Use this rule cautiously.

3. **Reciprocity Handling:** If a relationship idx_A → idx_B is established, also establish the reverse idx_B → idx_A, representing the bidirectional potential to use the identified key pair for joining.

4. **Consolidation:** For each source table index, collect all unique target table indices. Sort the target indices numerically.

**Output Format (Strict):** One line per source table index, starting from 0 up to N-1. Format: `SourceTableIndex:TargetIndex1,TargetIndex2,...`. No spaces. If a source table has no outgoing joins, the line must still be present (e.g., `5:`). The output must contain only these adjacency list lines with no additional text.

## Appendix B. Phrase-Schema Entity Mapping Prompt

**Role:** You are an expert Database Administrator specializing in Natural Language Processing (NLP) and Schema Mapping.
**Objective:** Analyze a given natural language QUERY and map meaningful words or phrases within it to the most relevant columns in the provided database TABLE INFORMATION.
**Inputs:**

1. QUERY: The natural language question posed by the user.

2. TABLE INFORMATION: A description of the database schema, including table names and column definitions (DDL or equivalent).

**Instructions:**

1. **Analyze Context:** Thoroughly examine both the QUERY and the TABLE INFORMATION. Understand the relationships between tables and the likely data stored in each column.

2. **Identify Mappable Terms:** From the QUERY, extract words or multi-word phrases that carry semantic weight related to the database schema. Focus on: Nouns (entities/attributes like "student", "name"); Verbs (actions like "find", "participate"); Adjectives/Adverbs only if they directly represent a state, category, or quantifiable attribute in a column (e.g., "oldest" → Age); Specific Values (like 'Smith', 123). Do not map adjectives/adverbs tied to operations (e.g., "different", "unique", "lowest").

3. **Perform Column Mapping:** For each identified term, determine its corresponding database column(s): Direct & Semantic Mapping (match terms to synonymous columns); Value Mapping (map specific values to the most likely `TableName.ColumnName`); Verb-to-Concept Mapping (map verbs to data columns reflecting results/status of the action); Multiple Mappings (list all valid mappings, one per line); N/A Mapping (if no match exists, map to N/A, used sparingly).

4. **Exclusions (DO NOT MAP):** SQL Keywords (`SELECT`, `FROM`, `WHERE`, etc.); Functions (`COUNT`, `SUM`, `AVG`, etc.); NL Phrases ("number of", "how many", "list all", etc.); General Modifiers ("different", "unique", "various", etc.); Stop Words ("what", "who", "the", "a", "in", etc.).

5. **Precision:** All mappings must be directly justifiable. No speculation.

**Output Format (strict):**

```
word_or_phrase - TableName.ColumnName
word_or_phrase - AnotherTable.AnotherColumnName
word_or_phrase_not_found - N/A
```

**Input:**

QUERY: {query}

Database INFORMATION: {Database Information}